41ª Jornadas Argentinas de Informática e Investigación Operativa

"LogicChess: Herramienta Didáctica para la Ejercitación en Lógica de Predicados de Primer Orden"

Autores: Arian Kiehr, Matías Ré Medina
e- mail: ariankiehr@gmail.com, aereal@gmail.com

Directores: **Mauco, Virginia; Felice, Laura; Ferrante, Enzo**
e- mail: *{vmauco, lfelice}@exa.unicen.edu.ar , eferrante@alumnos.exa.unicen.edu.ar*

Categoría: **Trabajos de Cátedra**

Área: **Ambientes de Programación, Algoritmos, Lenguajes, Modelos**
Lógica, Inteligencia Artificial, Sistemas Inteligentes

Asignaturas: **Ciencias de la Computación II, Análisis y diseño de algoritmos I**

2º año de la carrera de Ingeniería de Sistemas
Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires
Tandil

LogicChess: Herramienta Didáctica para la Ejercitación en Lógica de Predicados de Primer Orden

Resumen

LogicChess fue desarrollado como proyecto final correspondiente a dos materias de una carrera de Informática dictadas en el segundo año de la misma que introducen una, conceptos de análisis y diseño de algoritmos y la otra, un curso básico de Lógica. El objetivo fue crear una herramienta para el aprendizaje práctico de la Lógica de Primer Orden, haciendo énfasis en el análisis sintáctico y semántico de fórmulas para que el usuario (el alumno) pueda consolidar sus conceptos teóricos a través de un entorno amigable en el cual trabajar.

Básicamente el programa permite el ingreso de fórmulas lógicas y las evalúa sintácticamente para determinar si son fórmulas bien formadas, en caso de que lo sean podrán ser evaluadas semánticamente en algún modelo definido por el usuario para el *frame* soportado por la herramienta.

LogicChess ha sido diseñado de forma tal que pueda ser usado como apoyo en la ejercitación en cursos básicos de Lógica clásica. LogicChess es software libre, actualmente se encuentra liberado bajo la licencia GNU GPL v3.

1. Introducción

Los cursos básicos de lógica son un denominador común en la mayoría de las carreras de informática. En ellos, los estudiantes deben realizar gran cantidad de trabajo individual para la resolución de ejercicios, adquiriendo así práctica en el manejo de formalismos.

En este contexto, se gestó este proyecto con dos objetivos principales. Por un lado, es ampliamente aceptado que resulta útil la utilización de herramientas didácticas en cursos básicos de Lógica que acompañen el proceso de enseñanza/aprendizaje pero que no requieran excesivo tiempo de estudio para su utilización. Por otro lado, es crucial la integración del razonamiento teórico y empírico usando un enfoque metodológico de diseño y desarrollo del sistema, pudiendo de esta manera ser un aporte en ambos sentidos.

La herramienta desarrollada, LogicChess, es educativa, visual e interactiva. Permite a alumnos y docentes experimentar con Lógica de predicados de Primer Orden [3] resultando un buen suplemento didáctico. El sistema se ha diseñado bajo el paradigma de la Programación Orientada a Objetos, e implementado en lenguaje C++ y Java.

Además, ha sido liberado bajo la licencia GNU GPL v3 [2] siendo posible su descarga para uso y modificación del código fuente [1].

Se describe en la sección 2 el diseño global e implementación de la herramienta. La sección 3 describe los modelos sobre los cuales trabaja LogicChess, la sección 4 detalla la resolución general. Finalmente las conclusiones se encuentran en la sección 5.

2. Diseño global e Implementación

Para el diseño de la herramienta se tuvieron en cuenta los contenidos enseñados en el curso de lógica [4] y para el diseño e implementación, la metodología del curso de análisis y diseño de algoritmos [4]. Ambos cursos se dictan en el primer semestre del segundo año de una carrera de Informática.
Con respecto al modelo de diseño utilizado, se puede distinguir claramente tres niveles dentro de una arquitectura de niveles básica:

a. Nivel de Estructuras Fundamentales:

En este nivel se encuentran los tipos de datos abstractos (TDAs) para poder almacenar y trabajar con los elementos de las fórmulas de la Lógica de predicados de Primer Orden, siendo la base para los algoritmos que se implementarán en el nivel superior. Estos tipos de datos brindan una interfaz para el manejo de los elementos de la lógica e implementan algunos métodos básicos para la obtención de información potencialmente útil. Es aquí donde se vislumbra la relación entre los conceptos teóricos que hacen a la lógica y los párrafos de código fuente que los representan. La correspondencia establecida entre cada uno de los elementos lógicos y las clases que los implementan, hacen de este nivel el más rico en aspectos de diseño de estructuras.

Resulta de especial interés que la estructura de cada clase garantice un acceso eficaz a los datos que almacena, ya que de ello dependerá la complejidad de los algoritmos que se implementarán posteriormente.

Dada la modularización que se ha empleado en el diseño de las estructuras, es totalmente viable su utilización en otros programas a modo de librerías externas.

b. Nivel de Algoritmos

En este nivel se incluyen los algoritmos encargados de procesar los datos almacenados en las estructuras, convirtiéndolos en información útil para el usuario.

Los algoritmos fundamentales son: el que verifica la correcta sintaxis de una fórmula y el que comprueba si la fórmula es válida o no en el modelo dado. Estos algoritmos se implementaron en la especificacion del parser y en el tipo de dato Formula.

c. Nivel Gráfico

Para la implementación de este nivel, se utilizó el lenguaje Java. Aquí se almacena el código de la Interfaz Gráfica de Usuario (GUI) que brinda, sobre todo, facilidad y agilidad en la utilización del programa. Es posible utilizar la interfaz en cualquier sistema operativo sin la necesidad de compilar nuevamente gracias a las características del lenguaje. Además como el software es libre, si un alumno desea modificar parte del código que trata específicamente sobre la Lógica de predicados de Primer Orden, no necesita ninguna librería adicional de GUIs, ya que la parte del proyecto desarrollada en el lenguaje C++ fue diseñada totalmente para interactuar en modo consola e independiente del código Java. En la figura 1 se muestran los elementos principales de la herramienta como lo es el tablero y la ficha.

Figura 1. Elementos principales de la herramienta.

Con respecto a la implementación, LogicChess ha sido programado bajo el paradigma de la Programación Orientada a Objetos (POO) e implementado haciendo utilización del lenguaje C++ y Java. Se implementaron los tipos de datos abstractos (TDA) como

clases estructuradas, de manera tal que se mantenga la independencia entre los niveles de diseño descriptos anteriormente.

3. Modelos sobre los que trabaja LogicChess

LogicChess permite trabajar sobre un conjunto finito de modelos, por eso es importante detallar los modelos con los que trabaja para definir el alcance del lenguaje que comprueba esta herramienta. En este caso, cada modelo representa un tablero de Ajedrez. Este modelo está formado por Fichas, Números y Vacío (referencia a un casillero que no contiene fichas). Un número puede ser utilizado para representar una única coordenada o una distancia dentro del tablero. Si dos fichas están juntas la distancia entre ellas es 1. Cualquiera sea el caso (que represente una distancia o coordenada), un número deberá ser siempre menor estricto que 8 y mayor o igual a 0 ($0 <= N < 8$). A continuación se detallan los elementos del lenguaje que maneja LogicChess.

3.1 Funciones

- getFicha(numero, numero) *Dado un par de coordenadas retorna la ficha que se encuentra en esa posición o vacío si no hay ninguna ficha.*

3.2 Predicados

Se definen los predicados de *identificación*, los de *posición* y los de *distancia*.

- *identificación*

- esPeon(ficha) *Es válido si ficha es un Peón.*
- esAlfil(ficha) *Es válido si ficha es un Alfil.*
- esTorre(ficha) *Es válido si ficha es una Torre.*
- esCaballo(ficha) *Es válido si ficha es un Caballo.*
- esReina(ficha) *Es válido si ficha es una Reina.*
- esRey(ficha) *Es válido si ficha es un Rey.*
- esBlanca(ficha) *Es válido si ficha es Blanca.*
- esNegra(ficha) *Es válido si ficha es Negra.*
- estaVacia(numero, numero) *Es válido si en la posición no hay fichas.*

- *posición*

- mismaFila(ficha, ficha) *Es válido si las fichas están en la misma fila, sin importar distancia o si se interponen otras fichas.*
- mismaColumna(ficha, ficha) *Es válido si las fichas están en la misma columna, sin importar distancia o si se interponen otras fichas.*

- mismaDiagonal(ficha, ficha) *Es válido si las fichas están en la misma diagonal, sin importar distancia o si se interponen otras fichas.*
- estaEnL(ficha, ficha) *Es válido si las fichas están en L. Esto quiere decir que una se encuentre a dos casilleros de la otra, hacia arriba o hacia abajo y un casillero hacia los costados; hacia la izquierda o derecha y un casillero hacia arriba o abajo. Sin importar si se interponen otras fichas.*

- *distancia*

- distancia(ficha, ficha, numero) *Es válido si la distancia entre las 2 fichas es igual a numero. La distancia será evaluada en las diagonales, horizontales y verticales, si se pregunta por la distancia de dos fichas que no se encuentran en estas posiciones no será válido el predicado.*
- caminoLibre(ficha, ficha) *Es válido cuando entre las dos fichas no se interponga ninguna otra, ya sea en posición vertical, horizontal o en ambas diagonales.*

3.3 Representación de caracteres

El lenguaje que maneja LogicChess se corresponde a la Lógica de Primer Orden, en él existen: predicados, funciones, variables, constantes, operaciones, cuantificadores y paréntesis, sólo que a fines prácticos la sintaxis no es la misma. Se adaptó y restringió el alfabeto a las características de los posibles modelos que podemos representar. El alfabeto, en cuestión, está formado por:

Carácter	Representación
∃	#
∀	@
,	,
(	(
)	)
not	!
and	&
or	|
implica	>

Tipo de variables: [v][a-z0-9]

Ejemplos: va, vb, vc, ... , vz, v0, v1, ..., v9.

Tipo de Constantes (que representan fichas particulares): [**c][a-z0-9]**

Igual que las variables, se pueden utilizar todas las letras y números cualquiera del 0 al 9 siempre que se agregue la c primero.

Números: [0-7]

No puede haber un número mas alto que 7 dado que excede el tablero a la hora de usarlo dentro de getFicha(numero, numero), y también lo excede a la hora de preguntar por una distancia 8.

Definición de una fórmula:

Una de las principales características de LogicChess es poder indicar si una fórmula particular tiene o no errores sintácticos, resulta importante entonces explicar cuando una fórmula es correcta y cuando no lo es. La gramática que describe el lenguaje de LogicChess está descripta por el BNF que se presenta en la Figura 2.

```
<Sentencia> ::= <Sentencia Atomica>
                | <Sentencia Atomica> <Conector> <Sentencia>
                | <Cuantificador> <Variable> <Sentencia>
                | ! <Sentencia> | ( <Sentencia> )

<Sentencia Atomica> ::= <Pred_unario> ( <Ficha> )
                | <Pred_binario> ( <Ficha> , <Ficha> )
                | <Pred_dist> ( <Ficha> , <Ficha>, <Numero> )
                | <Pred_vacio> ( <Numero> , <Numero> )

<Ficha> ::= <Funcion>( <Numero> , <Numero> ) | <Constante> | <Variable>
<Numero>::= <Variable> | <Numero_term>
<Conector> ::= & | | | >
<Cuantificador> := # | @
<Constante> ::= ca | cb | cc | .. | c0 | c1 | c2 | .. c9
<Numero_term> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<Variable> ::= va | vb | vc | .. | v0 | v1 | v2 | .. | v9
<Pred_unario> ::= esPeon | esAlfil | esTorre | esCaballo | esReina | esRey | esBlanca
                | esNegra
<Pred_binario> ::= mismaFila | mismaColumna | mismaDiagonal | estaEnL
                |caminoLibre
<Pred_dist> ::= distancia
<Pred_vacio> ::= estaVacio
<Funcion> ::= getFicha
```

Figura 2. BNF correspondiente a la gramática del lenguaje de LogicChess.

En caso de que se cometa algún error sintáctico al escribir una fórmula, el sistema informa de dicha anomalía para que el usuario la corrija. También existe la

posibilidad de utilizar sintaxis alternativa si es que no se desea usar la que esta en el BNF, como por ejemplo:

- Para todo: PARATODO | PT
- Existe: EXISTE | EX
- Negación: NO | NOT | ¬
- Conjunción: AND | Y
- Disyunción: OR | O

Existen 3 tipos de respuestas por parte de la interfaz a la hora de evaluar una fórmula: **Errónea**, **Válida** y **Falsa**.
Errónea: Cuando la fórmula ingresada no pertenece al lenguaje descripto en la herramienta.
Falsa: Cuando la fórmula ingresada pertenece al lenguaje pero no es válida en el modelo dado.
Válida: Cuando la fórmula pertenece al lenguaje y es válida en el modelo.

4. Algoritmos para determinar la validez de una fórmula en un modelo

La resolución de una fórmula está ligada a la estructura que se utiliza para almacenarla. El TDA que se encarga de almacenar la fórmula es una clase abstracta *Fórmula* y hay tres clases que la heredan, las clases *OperacionCompuesta, OperacionSimple* y *Predicado*. En el caso de un predicado, este contiene información acerca del mismo, sus atributos (que pueden ser constantes, funciones, variables) y es capaz de poder evaluarse a sí mismo en el modelo dado. Cada predicado hace consultas diferentes al modelo (dependiendo que predicado sea y cuales sean sus atributos) y obtienen mediante este, su valor de verdad.

Las otras dos clases representan operadores y cuantificadores. Estas clases tienen la particularidad de que pueden contener en su interior (además del cuantificador u operador) otras fórmulas. La solución para poder representar esta estructura responde al patrón de diseño *composite*, que sugiere una estructura en forma de árbol y composición recursiva. Parte del diagrama de clases se muestra en la figura 4. Se puede encontrar el diagrama de clases completo del sistema en la documentación del proyecto, de manera online [5].

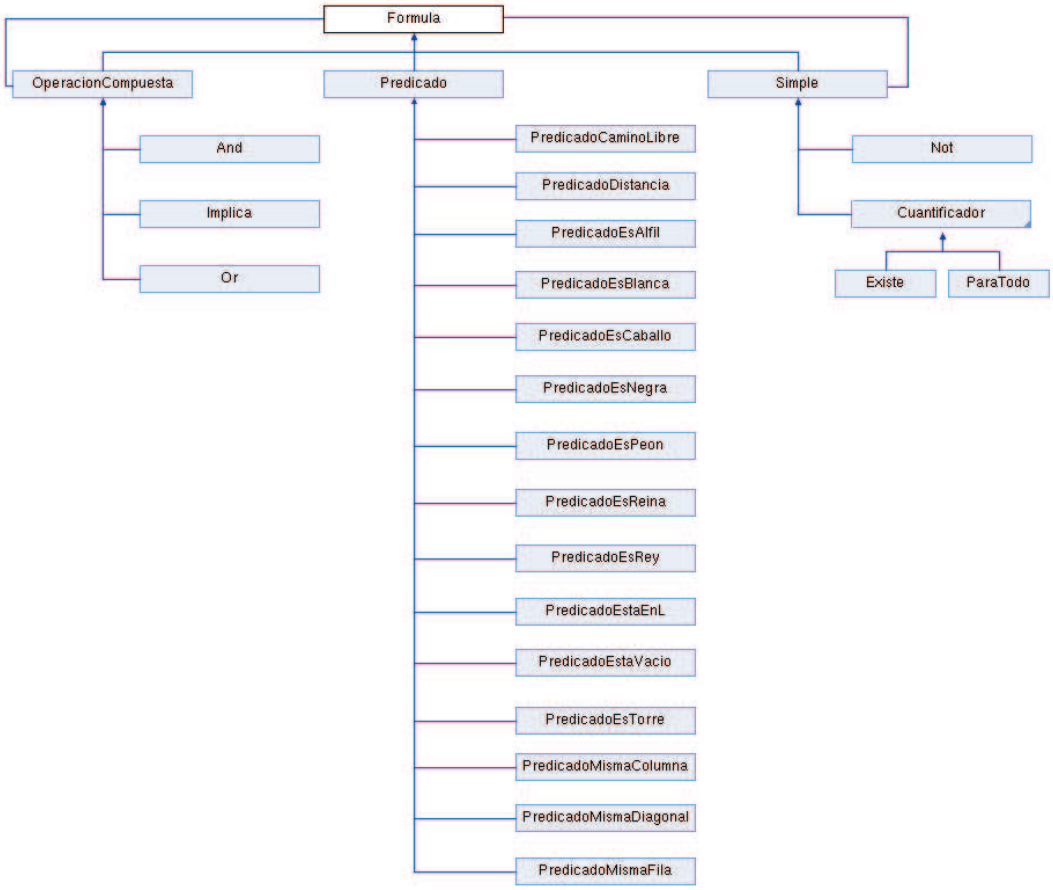

Figura 4. Diagrama de clases de Formula y sus clases derivadas

Una vez que se requiera el valor de una fórmula, el algoritmo invoca el método que calcula el valor (dicho método es abstracto en la clase padre pero cada clase derivada la implementa adecuadamente). En el caso de las operaciones compuestas, se retorna el valor de verdad de ambas fórmulas vinculados por el operador (por ejemplo si es un AND se retorna la conjunción lógica entre ambos valores de verdad). En caso de las operaciones simples, si es el operador unario NOT simplemente se retorna el valor de la fórmula negado y si es un cuantificador se remplazan todas las variables dentro de su alcance por cada uno de los elementos del dominio para que los predicados sepan con que valores evaluarse. Los predicados no se evalúan nunca con valores variables, estos son siempre remplazados por los cuantificadores.

A modo de ejemplo, si es un cuantificador existencial se realiza la disyunción lógica entre todos los valores que se calculan dentro de dicho cuantificador.

La figura 5 ilustra una resolución válida con cuantificadores.

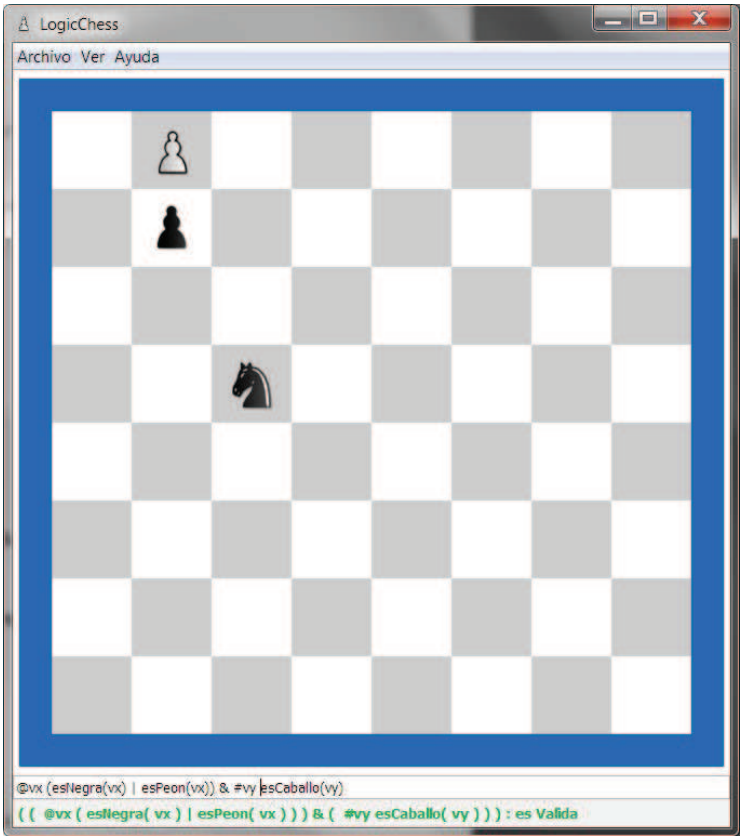

Figura 5. Resolución válida con cuantificadores.

5. Conclusiones

LogicChess fue pensado para la asistencia en el proceso de enseñanza / aprendizaje de un curso introductorio de Lógica. Fue desarrollado por estudiantes del segundo año de una carrera de Informática y actualmente se utiliza en las prácticas del curso mencionado siendo un complemento de uso muy ágil y amigable para jóvenes estudiantes de la carrera. Siendo software libre, brinda por ejemplo a otros estudiantes la posibilidad de utilizar y compartir sin restricciones los recursos que éste les ofrece y la posibilidad de permitir a los mismos explorar, modificar y analizar las implementaciones reales de los algoritmos propuestos. De esta forma, además de utilizarlas como mero instrumento de aprendizaje, el alumno puede ser partícipe de la construcción de sus propias herramientas y enriquecimiento de otras.

El diseño de LogicChess fue pensado para ser extensible. De este modo, se puede remarcar las funcionalidades que hacen falta en la versión actual del programa para implementar en una versión extendida. Una forma de extender la funcionalidad es que el sistema sea capaz de aceptar variables libres para permitir el uso de cualquier fórmula y no sólo fórmulas cerradas o sentencias (que es con lo que actualmente se trabaja).

Referencias

[1] MFSec/LogicChess - http://www.mfsec.com.ar/downloads/logicchess

[2] GNU licence - http://www.gnu.org/licenses/gpl.html

[3] Mathematical Logic for Computer Science. Ben-Ari, M. 2001 Springer Verlag

[4] Página del curso de Lógica y página del curso de Algoritmos.

[5] Documentación Online - http://www.mfsec.com.ar/downloads/logicchess/doc/

[6] Informe LogicChess presentado en cátedra. - http://www.mfsec.com.ar/downloads/logicchess/InformeLogicChess.pdf